**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #4>>**

**서강대학교 컴퓨터공학과**

**[학번] 20161643**

**[이름] 정진원**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 실행 방법
   2. 모델 구성
   3. Confusion Matrix
   4. Loss Function
   5. Rank
3. **코드** 
   1. 20161643.h
   2. 20161643.c
4. **프로그램 개요**

Google Colab에서 Keras를 이용하여 이미지 분류를 하는 머신 러닝 모델을 개발한다. CIFAR-10 데이터를 이용하여 모델을 train 하고, 파라미터들을 바꿔 정확도 80% 이상을 얻도록 하였다.

1. **프로그램 설명**
   1. **실행 방법**
2. sp20161643\_proj4.zip을 압축 해제한다
3. Google Colab을 실행한다.
4. 파일->노트 업로드->파일선택->20161643.ipynb 선택
5. 업로드 된 노트를 Ctrl + F9으로 실행시키면 모든 cell 들이 실행된다.

* 1. **모델 구성**

-기존의 모델보다 정확도를 올리기 위해 layer들을 추가해줬다. Convolutional layer의 input과 output은 4D array의 형태를 가진다. input data는 (batch\_size, height, width, depth)를 형태로 가진다. First dimension은 batch size를 나타내고, 나머지 세개의 dimension은 이미지를 나타낸다. width와 height는 이미지의 크기를 나타내고 depth는 color channel의 number를 나타낸다. RGB image의 경우 color channel의 number는 3이된다. 이는 도식화한 layer에 (None, 32, 32, 3)의 형태로 나타난다. 이후 필터의 개수에 따라 depth가 변화한다. Dense layer의 경우 2D array가 input이어서 flatten layer를 추가해서 4d output을 2d 형태로 바꿔주었다.

-Convolution 과정을 거치면서 input 이미지의 크기가 줄어드는 것을 방지하기 위해 padding을 추가해주었다.

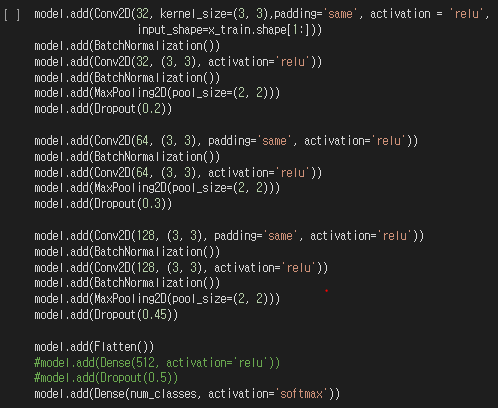
-activation 방법은 relu를 사용했는데, 조사해본 바로는 relu가 일반적이고 elu와 다른 activation 방법들과 정확도에 큰 차이가 없어서 relu를 선택하였다.

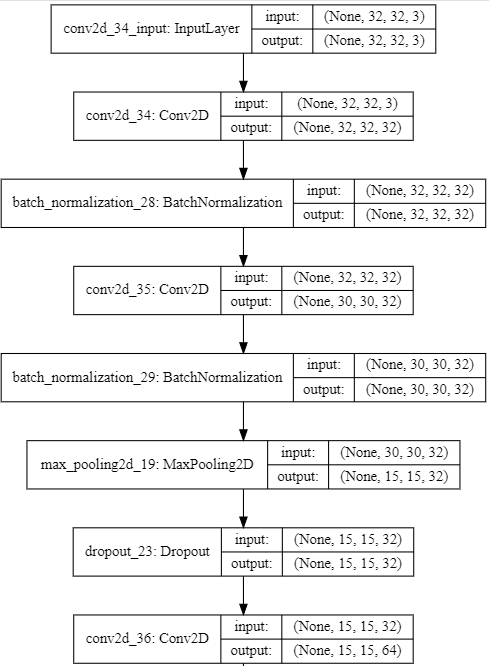
-중간중간 BatchNormalization을 넣어줬는데, activation 전에 normalization을 하는 것이 일반적이지만 시간이 많이 증가하여 activation 이후에 넣어주었다. 정규화는 다음 레이어에 일정한 분포를 가진 입력을 제공하여, 학습이 잘 이루어지도록 한다.

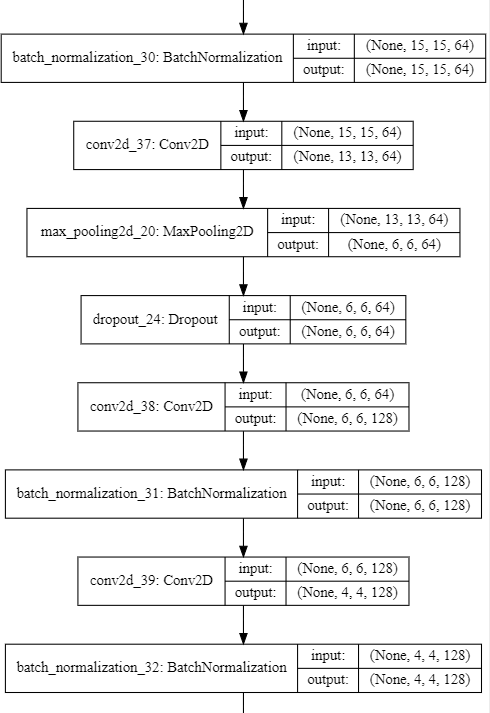
-Dropout도 추가하여 overfitting이 덜 나타나도록 하였다. 수치는 값들을 바꿔가며 결과가 좋은 값을 채택하였다. dropout rate이 너무 높은 경우 학습을 제대로 하지 못했다.

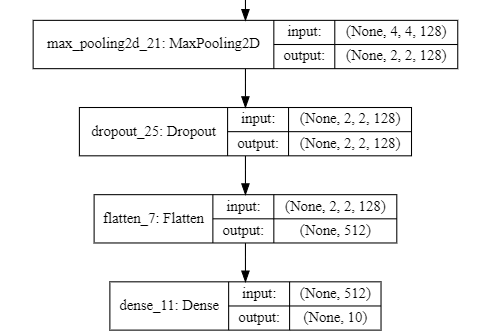
-learning rate을 0.01에서 0.001로 바꾸었는데 accuracy가 많이 증가하여 learning rate을 낮춰주었다. learning rate가 너무 높은 경우 최적의 값으로 수렴하지 않고 발산해버리는 overshooting이 일어날 수 있다.

-optimizer는 rmsprop을 사용하였는데, 평균적으로 SGD 보다 적은 epoch으로 높은 accuracy에 도달하여 선택했다.



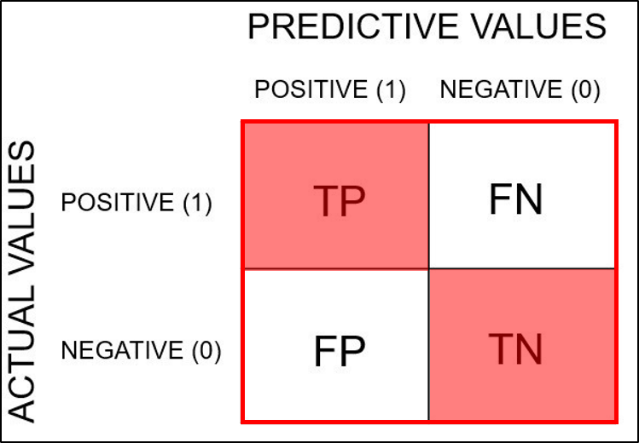




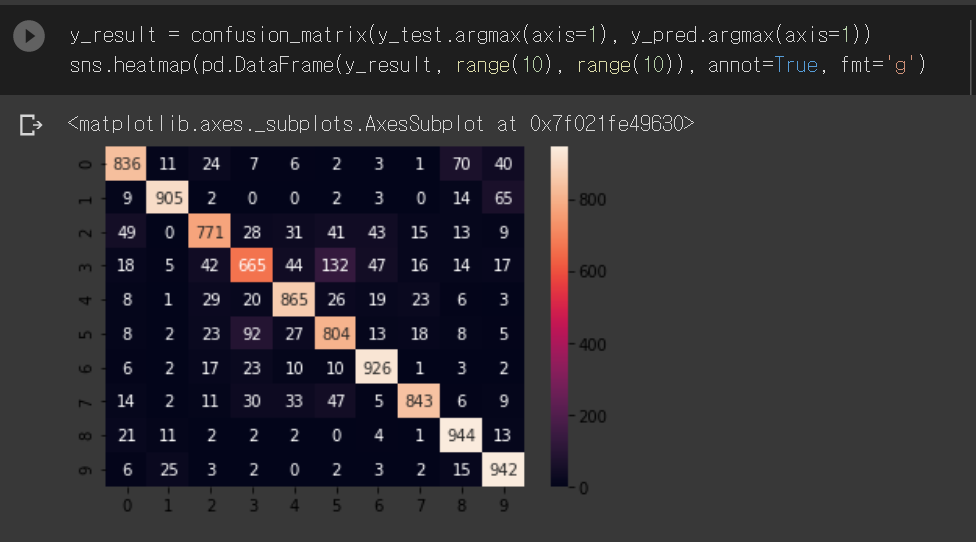


* 1. **Confusion Matrix**

10개의 클래스에 대해 모델의 분류 결과를 확인 할 수 있다. Training을 통한 prediction 성능을 측정하기 위해 예측 value와 실제 value를 비교하기 위한 표이다. 다음 그림과 같이 나타내며 T는 true, F는 false, P는 positive, N은 negative를 의미한다. 이중 모델이 바르게 분류한 부분은 confusion matrix의 대각선 부분이다.

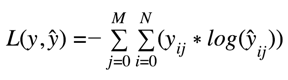






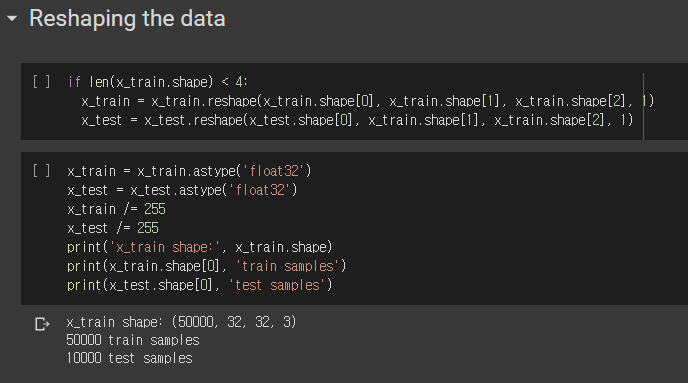
* 1. **Loss Function**

Categorical Cross Entropy loss가 사용되었다. Softmax activation 뒤에 cross entropy loss를 붙인 형태로 주로 사용하기 때문에 Softmax loss라고도 불린다. 여러 개의 분류 결과 중 하나의 결과만 참이 될 수 있는 경우에 사용된다.



y hat은 predicted value이다. Categorical cross entropy는 예측 결과의 distribution과 (activations in the output layer, one for each class) probability가 0, 1로 set 된 true distribution을 비교한다. 다시 말해 true class는 one hot encoded vector로 나타나고, model의 결과가 이 벡터와 가까울수록 loss가 작아진다.

* 1. **Rank**



학습 데이터와 트레이닝 데이터의 CNN의 input으로 사용되기 위해 rank가 4 여야 한다. 따라서 전처리 과정에서 x\_train.shape의 rank가 4 이하이면 reshape을 통해 rank를 4가 되게 만들어주었다. x\_train.shape을 출력해보면 rank가 네개인 것을 확인할 수 있다.

1. **코드** 
   1. **20161643.ipynb**

*# -\*- coding: utf-8 -\*-  
"""05.20161643.ipynb  
  
Automatically generated by Colaboratory.  
  
Original file is located at  
 https://colab.research.google.com/drive/1TXQwkqom2yym8n14M7GqR-aPsaKtiI2E  
  
# Import modules  
"""  
  
# Commented out IPython magic to ensure Python compatibility.***import** keras  
**from** keras.datasets **import** cifar10  
**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense, Dropout, Flatten, Activation  
**from** keras.layers **import** Conv2D, MaxPooling2D  
**from** keras **import** backend **as** K  
**from** keras.utils.vis\_utils **import** model\_to\_dot  
**from** IPython.display **import** SVG  
*# %matplotlib inline***import** matplotlib.pyplot **as** plt  
**from** sklearn.metrics **import** confusion\_matrix  
**import** pandas **as** pd  
**import** seaborn **as** sns  
**from** keras.layers **import** BatchNormalization  
  
**"""# Tuning parameters"""**epochs = 50  
learning\_rate=0.001  
  
batch\_size = 128  
num\_classes = 10  
  
**"""# Data  
  
## Plot image  
"""  
  
def** plot\_images(x, y\_true, y\_pred=**None**, size=(5, 5)):  
 **assert** len(x) == len(y\_true) == size[0] \* size[1]  
   
 fig, axes = plt.subplots(size[0], size[1])  
 fig.subplots\_adjust(hspace=0.5, wspace=0.1)  
  
 **for** i, ax **in** enumerate(axes.flat):  
 **if** x[i].shape[-1] == 1:  
 ax.imshow(x[i].reshape(x[i].shape[0], x[i].shape[1]))  
 **else**:  
 ax.imshow(x[i])  
  
 **if** y\_pred **is None**:  
 xlabel = **"True: {0}"**.format(y\_true[i].argmax())  
 **else**:  
 xlabel = **"True: {0}, Pred: {1}"**.format(y\_true[i].argmax(),   
 y\_pred[i].argmax())  
  
 ax.set\_xlabel(xlabel)  
   
 ax.set\_xticks([])  
 ax.set\_yticks([])  
  
 plt.show()  
  
**"""## Load dataset"""**(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()  
  
**"""## Reshaping the data"""  
  
if** len(x\_train.shape) < 4:  
 x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1)  
 x\_test = x\_test.reshape(x\_test.shape[0], x\_train.shape[1], x\_train.shape[2], 1)  
  
x\_train = x\_train.astype(**'float32'**)  
x\_test = x\_test.astype(**'float32'**)  
x\_train /= 255  
x\_test /= 255  
print(**'x\_train shape:'**, x\_train.shape)  
print(x\_train.shape[0], **'train samples'**)  
print(x\_test.shape[0], **'test samples'**)  
  
**"""## Applying One hot encoding for the data"""**y\_train = keras.utils.to\_categorical(y\_train, num\_classes)  
y\_test = keras.utils.to\_categorical(y\_test, num\_classes)  
  
**"""## Show data"""**plot\_images(x\_train[:25], y\_train[:25])  
  
**"""# Creating the DNN model"""**model = Sequential()  
  
**"""## Adding layers to the model"""**model.add(Conv2D(32, kernel\_size=(3, 3),padding=**'same'**, activation = **'relu'**,  
 input\_shape=x\_train.shape[1:]))  
model.add(BatchNormalization())  
model.add(Conv2D(32, (3, 3), activation=**'relu'**))  
model.add(BatchNormalization())  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.2))  
  
model.add(Conv2D(64, (3, 3), padding=**'same'**, activation=**'relu'**))  
model.add(BatchNormalization())  
model.add(Conv2D(64, (3, 3), activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.3))  
  
model.add(Conv2D(128, (3, 3), padding=**'same'**, activation=**'relu'**))  
model.add(BatchNormalization())  
model.add(Conv2D(128, (3, 3), activation=**'relu'**))  
model.add(BatchNormalization())  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.45))  
  
model.add(Flatten())  
*#model.add(Dense(512, activation='relu'))  
#model.add(Dropout(0.5))*model.add(Dense(num\_classes, activation=**'softmax'**))  
  
**"""## Visualization the model"""**SVG(model\_to\_dot(model, show\_shapes=**True**, dpi=70).create(prog=**'dot'**, format=**'svg'**))  
  
**"""## Optimizer"""**optimizer = keras.optimizers.rmsprop(lr=0.001,decay=1e-6)  
  
**"""## Compiling the model"""**model.compile(loss=keras.losses.categorical\_crossentropy,   
 optimizer=optimizer,   
 metrics=[**'accuracy'**])  
  
**"""## Training the model"""**model.fit(x\_train, y\_train,  
 batch\_size=batch\_size,  
 epochs=epochs,  
 verbose=1,  
 validation\_data=(x\_test, y\_test))  
  
  
  
**"""## Evaulating the model"""**score = model.evaluate(x\_test, y\_test, verbose=0)  
print(**'Test loss:'**, score[0])  
print(**'Test accuracy:'**, score[1])  
  
**"""## Prediction the model"""**y\_pred = model.predict(x\_test)  
  
plot\_images(x=x\_test[:25], y\_true=y\_test[:25], y\_pred=y\_pred[:25])  
  
y\_result = confusion\_matrix(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))  
sns.heatmap(pd.DataFrame(y\_result, range(10), range(10)), annot=**True**, fmt=**'g'**)